Synchronous byzantine agreement with nearly a cubic number of communication bits

Dariusz R. Kowalski, Achour Mostefaoui

Synchronous Byzantine Agreement with Nearly a Cubic Number of Communication Bits

Dariusz R. Kowalski[†] and Achour Mostéfaoui[⋆]
[†] Department of Computer Science, University of Liverpool, UK
[⋆] IRISA/ISTIC, University of Rennes 1, France
D.Kowalski@liverpool.ac.uk achour@irisa.fr

Abstract

This paper studies the problem of Byzantine consensus in a synchronous message-passing system of n processes. The first deterministic algorithm, and also the simplest in its principles, was the Exponential Information Gathering protocol (EIG) proposed by Pease, Shostak and Lamport in [19]. The algorithm requires processes to send exponentially long messages. Many follow-up works reduced the cost of the algorithm. However, they had to either lower the maximum number of faulty processes t from the optimal range $t < n/3$ to some smaller range of t [4, 11, 18], or increase the maximum worst-case number of rounds needed for termination (the lower bound being $t + 1$) [3, 9, 20].

Garay and Moses [13] were the first and only who solved the problem by using a polynomial number of communication bits, for the whole optimal range $t < n/3$ of the number of Byzantine processes and within the optimal number $(t + 1)$ of communication rounds. Their solution, though very complex and sophisticated, requires processes to send $O(n^9)$ bits in total.

In this work, we present much simpler solution that also holds for the whole optimal range $t < n/3$ and the optimal number $t + 1$ of communication rounds, and at the same time lowers the number of exchanged communication bits to $O(n^3 \log n)$. For achieving such an improvement, processes no more exchange relayed proposed values, but information on suspicions "who suspects who", the size of which is quadratic in n in the worst case.

Keywords: Agreement problem, Byzantine process, Consensus, Synchronous distributed system, Message-passing model, Round-based protocol, EIG.

1 Introduction

The Consensus problem is considered as a fundamental problem in fault-tolerant distributed systems. In case processes can exhibit a malicious behavior, the obtained variant of consensus is often called Byzantine Agreement. In order to assure perfect reliability, deterministic solutions are of utmost importance for this problem, and this is also the focus of this paper; for recent advances and references to the area of randomized solutions we refer the reader to the work by King and Saia [15] and by Aspnes [1].

The first and the simplest designed deterministic algorithm for the synchronous distributed computing model is the exponential information gathering protocol (from now on called EIG protocol) based on a tree construction proposed by Bar-Noy, Dolev, Dwork and Strong [3] as a simple

reformulation of the original protocol proposed by Pease, Shostak and Lamport in 1980 [19]. The algorithm requires processes to send exponentially long messages and performs exponentially many local computation steps, in terms of the number n of processes. Many researchers have since tried to reduce the cost of the solution, However most of them either lowered the maximal range of the tolerated number of faulty processes t from $t < n/3$ (the upper bound) to some smaller values [11, 18, 4] or increased the maximum number of rounds needed in the worst case to terminate (the lower bound being $t + 1$) [3, 9, 20]. For such a purpose, they developed a series of techniques to exploit the redundancy encountered in the tree structure of the EIG protocol, sacrificing the optimum fault-tolerance or the minimum number of communication rounds. [3, 7] propose a trade-off between the number of rounds and the bit complexity for any constant d they increase the number of rounds to $t+t/d$ while decreasing the bit complexity to $O(n^d)$. Similarly, [4] proposes a trade-off between the resilence of the protocol and the bit complexity for any constant ϵ the ration of faulty processes is increased to $(3 + \epsilon)t$ while the bit complexity is lowered to $poly(n) \cdot O(2^{1/\epsilon})$.

Garay and Moses [13] were the first and only who solved the problem with polynomial number of communication bits and polynomial local computation cost, still meeting both the bound of $t < n/3$ on the number of Byzantine processes and the number of $t+1$ communication rounds. Their solution is very complex, and both polynomials are actually large — according to our estimates they are both $O(n^9)$ — thus leaving a huge space for improvement. The only known lower bound $\Omega(nt)$ on the number of communication bits was proved by Dolev and Reischuk [10]; this bound becomes $\Omega(n^2)$ for linear number of Byzantine processes. The bound $t < n/3$ on the number of Byzantine processes necessary for reaching consensus was given by Lamport, Shostak and Pease [16]. Fisher and Lynch [12] proved the lower bound $(t + 1)$ on the number of communication rounds, which holds even for milder process failures such as crashes. The interested reader can find in [14] the history of the improvements introduced from the initial protocol in 1980 [19] to the polynomial one in 1998 [14] (the full version of [13]). We are not aware of any improvements since then.

The table below taken from [13] gives an overview of the most important improvment over time concerning the consensus problem. Other improvments occured but they consider randomization; King and Saia [15] reduced the bit complexity to sub-quadratic.

Protocol	n	rounds	comm. bits
PSL 1980 [19]	$3t + 1$	$t + 1$	$\exp(n)$
DFFLS,TPS 1982 [9, 20]	$3t + 1$	$> 2t$	$poly(n)$
C 1985 [7]	$4t + 1$	$t + t/d$	$O(n^d)$
BD,DRS 1986 [2, 11]	$\Omega(t^2)$	$t + 1$	$poly(n)$
BDDS 1987 [3]	$3t + 1$	$t + t/d$	$O(n^d)$
MW 1988 [18]	$6t + 1$	$t + 1$	$poly(n)$
BGP 1989 [5]	$4t + 1$	$t + 1$	$poly(n)$
BG 1991 [4]	$(3 + \epsilon)t$	$t + 1$	$poly(n)O(2^{1/\epsilon})$
GM 1993 [13]	$3t + 1$	$t + 1$	$O(n^9)$
This paper	$3t + 1$	$t + 1$	$O(n^3 \log n)$

Our result In this work, we present a solution that is simpler yet more efficient in terms of communication complexity than the best known deterministic solution by Garay and Moses [14]. In particular, it matches the two bounds: $t < n/3$ on the number of tolerated Byzantine processes, and $t + 1$ on the number of communication rounds, while the total number of communication

bits $O(n^3 \log n)$ sent by non-Byzantine processes is close by factor $O(n \log n)$ to the lower bound $\Omega(n^2)$ on the total number of communication bits. In order to achieve this, the processes do not exchange the values they received in the previous rounds, but instead, they exchange a "digest" on the information they received. Based on these digests, they locally maintain information of "who suspects who" during each round. Thanks to this technique, the size of the exchanged information much reduced compared to former works, and the crucial part of the analysis shows that this very limited information is still sufficient to "mimic" the evaluation procedure done by the original EIG protocol "in spirit". It needs to be said that the evaluation made by our algorithms, based on very limited information exchange, might be slightly different than the evaluation made by the original EIG algorithm executed against the same adversarial schedule, but similar mechanisms guarantee agreement and validity of the final evaluation throughout correct nodes.

EIG protocol vs. our approach The EIG protocol works in two phases. During the first phase, processes exchange messages during $t+1$ communication rounds. At each round each process forwards all the information it received in the previous round to all processes. This protocol can be seen as the fully informed protocol, and hence it results in super-exponential communication complexity (understood as the number of exchanged bits). The collected information is stored in tree-like data structures, to which we refer as *evaluation trees*. At the end of round $t + 1$, the second phase (local computation on the local trees) is started. It consists of a bottom-up computation on each of the collected trees (one per process), where a resolved value is associated to each node of the tree structure by applying a resolution function to the children of this node. The decision value of the process is the resolved value associated with the set of roots of the trees. The polynomial protocol by Garay and Moses [13] uses the same tree structures, though parts of the trees are cut using sophisticated techniques. One of the introduced techniques consists in detecting cheating processes (Byzantine processes) in a given round in order to prevent them from cheating in subsequent rounds.

Our protocol is similar to the EIG protocol in its way of proceeding, as processes exchange information during $(t + 1)$ rounds and store the information they receive in local data structures. The fundamental difference is in the nature of the stored data. Instead of storing proposed values and then arrays of received values, arrays of arrays of received values, etc., the processes exchange only a relatively small digest of received information. More precisely, each process manages an array of process ids of size n, where it stores the ids of processes it itself suspects to be Byzantine. At the end of each round this information is updated and the new array is sent to all processes in the next round. Moreover, in order to detect cheating processes, a classical confirmation mechanism is used. All information broadcast by a process during a round is echoed by each receiving process to all other processes (the same mechanism is used for the consistent broadcast of Bracha and Toueg [6]). A message sent at a round r is confirmed at some process p_i at the end of round $r + 1$ and its sender in consequently not suspected if the same copy is relayed to p_i by at least $(n - t)$ processes during round $r + 1$. Conversely, if the message of some process is not confirmed in the next round, the sender is suspected. The fact that processes exchange suspicion arrays prevents the size of the exchanged information from exponential growth. To ease the presentation of our algorithm, we assume that the suspicion information is stored in a data structure similar to the EIG tree, although the size of the information sent by each correct process during each round is

only quadratic in the number n of processes.[1]

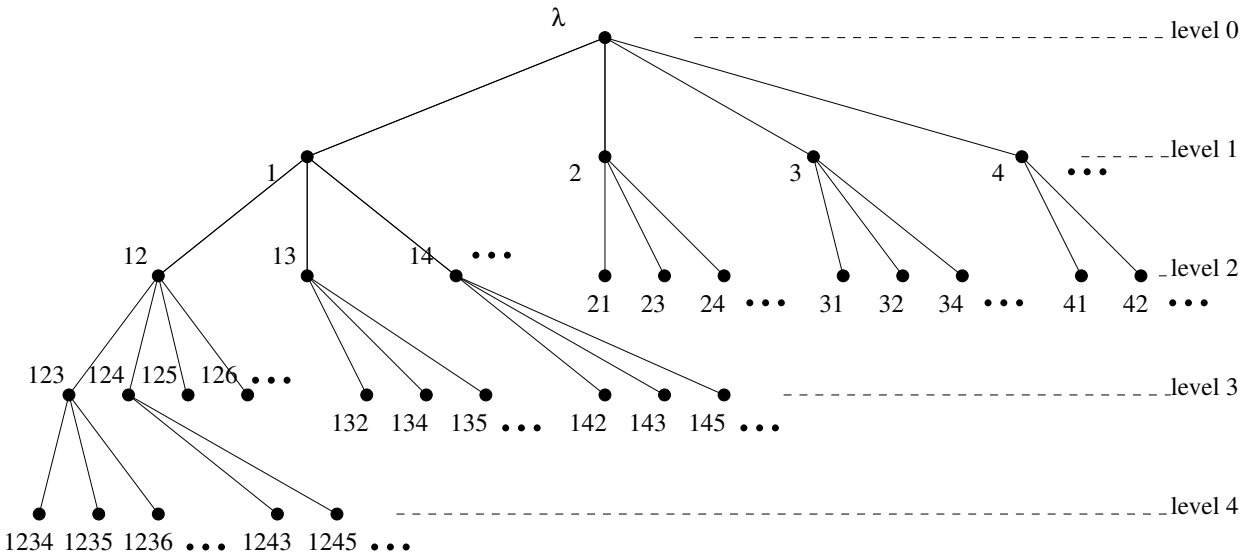

Figure 1: An EIG tree

In the remainder, we proceed as follows. In Section 2 we introduce the model and the consensus problem in more details. Section 3 presents the EIG tree structure used by the EIG original protocol and almost all other protocols that meet the $t < n/3$ bound. Section 4 describes the main algorithm. The analysis of the algorithm is split into two sections: the correctness proof of the protocol is given in Section 5, and the complexity measures are given in Section 6.

2 Byzantine Agreement and Computation Model

In this paper, we consider the classical Byzantine distributed synchronous message-passing model. The system is composed of a set Π of n synchronous processes, with ids in $\{1, \ldots, n\}$, which communicate through a reliable synchronous point to point channels (there is an a priori known bound on both message transfer delays and local computation of processes). Moreover, up to t processes can exhibit a *Byzantine* behavior, which means that such a process can behave in an arbitrary manner. This is the most severe process failure model: a Byzantine process can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, send different values to different processes, perform arbitrary state transitions, etc. A process that exhibits a Byzantine behavior during an execution is called *faulty*. Otherwise, it is called *correct*.

In the *Byzantine Agreement* problem, also called *Consensus*, each process p_i proposes an initial value v_i (this can be any value). Contrarily to many previous works including the only polynomial solution of Garay and Moses [14] that restrict the set of proposable values to $\{0, 1\}$ (called binary consensus, only two different values are proposable), we consider multivalued consensus.

[1] This suggests that the bit complexity of the algorithm is $O(n^4)$, as each of the n processes sends up to n^2 bits during each of the $t < n/3$ rounds of the algorithm; however we will show later in Section 6 that this amount of information can be reduced to $O(n^3 \log n)$.

The goal for each (correct) process is to decide on a value. A Byzantine agreement protocol has to satisfy the following three properties:

In this paper we consider deterministic synchronous round-based algorithms. Moreover, we do not use authentication (public key asymmetric cryptography to sign messages). During a round, each correct process may send to all processes, including itself, a unique message whose content is specified by the executed algorithm, and all messages are received within the same round. Then each process performs some local computation before starting a new round, or it decides and terminates.

The efficiency of algorithms is measured by the number of communication rounds (optimally $t+1$), and the number of bits sent in messages by correct processes (for which the known lower bound is $\Omega(n^2)$). Fault tolerance of algorithms is measured by the number of Byzantine processes that can be tolerated — ideally any number $t < n/3$.

3 The EIG Tree Structure

The exponential information gathering tree (EIG tree) is a data structure used to recording the information received by a given process along the successive rounds of an n-process round-based distributed computation. This structure is costly to maintain, as its number of nodes is exponential in the number n of processes. The trees we consider have (at the end of the $t+1$ rounds of the execution) exactly $t+2$ levels (from 0 to $t+1$).[2] A node of level ℓ has exactly $n-\ell$ children and has a label that consists of an ℓ element string of process ids (a process id appears at most once in a given label). The root of this tree is labeled with an empty string λ. The set of labels of the EIG tree is the set of all possible strings of size at most $t+1$, hence the size of the tree. Figure 1 presents an example of an EIG tree associated with some process p_i. We consider in this example a number of processes n greater than 5 but we cannot represent the whole tree. One can see that starting from level 1, and for the sake of clarity, many nodes are not represented and are replaced by $(\cdots)$.

During the execution of the original EIG protocol, each node x of the tree is used to store two values: $val(x)$ and $newval(x)$. The value of $val(x)$ of a node x on level ℓ is assigned during round ℓ. It is the ℓ-th hand report of some initial value (i.e., received through the chain of processes represented by the ids constituting the label of node x). The value $newval(x)$ of a node x is assigned during the decision procedure after the $t+1$ rounds.

4 The Main Algorithm

The proposed protocol proceeds in $t+1$ synchronous rounds, similarly to the EIG protocol. During a round r, each process sends a same message to all other processes (sending phase), next it waits for the messages sent to it (reception phase), and then it does local computation on both received and locally stored data. The fundamental difference between our solution and the EIG algorithm, as mentioned earlier, is in the nature of the stored and sent data. In other words, we use the same EIG tree structure that is decorated in a different way. Instead of storing the proposed values and then

[2]Sometimes an EIG tree was presented as a collection of n trees, each of $t+1$ levels ranging from 1 to $t+1$; these trees correspond to the n subtrees of our single EIG tree, each rooted in a distinct node at level 1. Both these approaches are equivalent.

the arrays of received values, the arrays of arrays of received values, etc., kept in the exponential size EIG tree structure, the processes maintain and exchange only a digest of received information. More precisely, during the first two rounds, processes exchange proposed values (similarly to the EIG protocol). Then, starting from the third round, each process informs the other processes which processes it is sure are Byzantine.

To ease the presentation of our algorithm, we say that a correct process p_c *received a default value* $\perp$, which cannot be proposed to the consensus, from a Byzantine process p_b, when process p_c received no value, or it received a non well-formed message, or if it has itself suspected p_b as being Byzantine in a previous round.

The two basic ideas that underlie and distinguish the proposed approach from previous works are: the exchange of suspicion information (except for the first two rounds) instead of the proposed values or the whole history of messages, and the use of the echo mechanism (here, called a confirmation mechanism) introduced by Bracha and Toueg [6] for asynchronous systems. A message can be safely delivered (we say here that it is *confirmed*) if it has been echoed by at least $(n-t)$ different processes. Bracha and Toueg proved that this mechanism ensures that if a correct process confirms a value from a given process at some round, no other correct process can confirm a different value from the same process at the same round (either it confirms the same value or no value at all). Moreover, values broadcast by correct processes are always confirmed (safely delivered) by at least all correct processes. They proved that in order to implement this mechanism, it is necessary to have $t < n/3$.

During a round r, $r \geq 3$, each process broadcasts a message that contains two parts. The first part is the data that the process wants to disseminate, called *main information* or *main part* of a message, and the second part consists of the main information the process has received from each process in the previous round. This second part is called the *proof* by Bracha and Toueg [6]; we call it the *echo*, due to its nature, and it will serve to confirm the main information of the previous round. A main information received from a given process p_j at round r is *confirmed* to a process p_i at round $r+1$ if it has been echoed to p_i (i.e., forwarded in the echo part of messages at round $r+1$) by at least $(n-t)$ different processes.

In the algorithm description below, we only specify the behavior of correct processes, as Byzantine processes, due to their malicious nature, are not bounded by the rules of the algorithm.

4.1 Principles of the Algorithm

Each process maintains two data structures: an EIG tree to store received values, and a set *byz* (initialized to $\emptyset$) to store the ids of the processes it suspects to be Byzantine (it suspects by itself, not as reported by some other process). Similarly to the original EIG protocol, each node of the EIG tree with label x can store values (noted $val(x)$ and $newval(x)$ in the original EIG protocol). In the present paper, each node stores *three*: values $val(x)$ and $cval(x)$, representing respectively the main part and the echo part of messages, and $newval(x)$ assigned to node x in the decision procedure at the end of round $t+1$. $newval(x)$ can be seen as a correction of $val(x)$. The values $val(x)$ and $cval(x)$ are collected during the different rounds, and $newval(x)$ is set during the extraction of the decision value after the $t+1$ rounds. In the presentation of the algorithm, when we say that we *decorate node* x, we mean that the variables $val(x)$ and $cval(x)$ associated with the node labeled x are set to some values (except of nodes x on level 1, for which $cval(x)$ is not defined).

The following algorithm describes the execution of the protocol from the point of view of a correct process p_i and shows how this process manages its two data structures: the EIG tree and the set byz_i.

Round 1. Each process sends its initial value to all other processes; this is the main part of the message and there are no echoes in this round. The values received by process p_i are used to decorate the nodes of *level* 1 of the EIG tree maintained by p_i. As explained above, if p_i has not received a message, or received a non well-formed message from a given process p_x, it decorates the node labeled x with the value $\perp$, i.e., $val(x) \leftarrow \perp$. If it received a proposable value v, it sets $val(x)$ to v. The variable $cval(x)$ is not used for nodes x on level 1.

Round 2. Each process p_i sends an empty main part and echoes the messages (i.e., initial values) it received in the first round. Thus, each message is a sequence of up to n initial values. The values received by p_i (up to $n(n-1)$ values) are used to decorate the nodes of level 2 of the EIG tree maintained by p_i. Node $x = jk$ is decorated by the value v, i.e., $cval(x) \leftarrow v$, if p_k reports that it received v from p_j at level 1; $cval(x)$ is set to $\perp$ if no valid value is reported. The variable $val(x)$ will be set during the next round. Moreover, during the computation phase of round 2 (after the confirmation checking of the messages sent during the first round), a correct process p_i adds to its set byz_i of suspected Byzantine processes any process p_b whose initial value (the one it received directly from p_b at round 1) is not confirmed at round 2.

Round 3. Each process p_i attaches its local set byz_i of Byzantine processes as the main part of its message (the "digest"), and echoes the arrays of initial values it received during round 2 (i.e., up to n arrays of n values each).

Here we can see a difference from the original EIG protocol. During a round r, $3 \leq r \leq t+1$, a correct process not only decorates the nodes of level r but also possibly the nodes from level $r-1$ to level $t+1$. Indeed, the confirmation mechanism needs two rounds to detect Byzantine behaviors and once we uncovered a Byzantine process, of course we never rely on what she says. For example, if p_j behaved in a Byzantine way (e.g., sending different values to different correct processes) during round 1, it will be suspected by at least one correct process p_k at the latest at the end of round 2 (using the confirmation mechanism). Then, p_k sends the information "p_k suspects p_j" to the other processes at round 3. We will see below how the variables $val(x)$ and $cval(x)$ associated with the different nodes x are set. Finally, p_i enriches its list of Byzantine processes using the confirmation mechanism applied to the messages received in the previous round and echoed in this round.

Round r, $4 \leq r \leq t+1$. Starting from round 4, rounds are the same as for round 3, except that the messages sent by the different processes include the list of Byzantine processes as the main part of the message, and echo the lists received during the previous round in the echo part (the proposed values are no more sent). We will define below how the variables $val(x)$ and $cval(x)$ associated with the different nodes x are set in these rounds.

Once the $t+1$ rounds are terminated, the EIG tree is decorated as follow. Recall that we already showed how to define $val(v)$ for any node x on level 1 of the tree, and also the values $cval(x)$ for nodes on level 2. In the following, we decorate the EIG tree with two values $\top$ and $\perp$. The value $\top$

means "do not suspect" whereas the value $\perp$ means "suspect". For example, when $val(x)$ of some node $x = yjk$ is set to $\top$, this means that p_k reported that it suspects p_j, and when $cval(x)$ of some node $x = yjk\ell$ is set to $\top$, this means that p_ℓ reported that p_k does not suspect p_j (this is a second-hand information — an echo).

- For each node x from a level greater than 1 and at most t, $x = yk\ell$ where y is a string (possibly empty) of ids and k, ℓ are ids of two other processes not listed in y: p_i sets $val(x)$ to $\top$ if p_ℓ never reported to p_i by round t that it suspects p_k; otherwise, $val(x)$ is set to $\perp$.
- For each node x from a level greater than 2, $x = yjk\ell$ where y is a string (possibly empty) of ids and j, k are ids of two other processes not listed in y: p_i sets $cval(x)$ to $\top$ if p_ℓ never reported by round $t + 1$ to p_i that p_k reported that it suspects p_j (this value is reported in the second-hand information included in the echo part of the message received by p_i from p_ℓ); otherwise, $cval(x)$ is set to $\perp$.
- One can note that for the nodes of level $t + 1$ (i.e., the leaves of the EIG tree), the variable $val(x)$ is set to no value, so for each leaf node x, $x = yk\ell$ where y is a string (possibly empty) of ids and k, ℓ are ids of two other processes not listed in y: p_i sets $val(x)$ to $\top$ if p_ℓ did not report to p_i that it suspects p_k; otherwise, $val(x)$ is set to $\perp$.

Remark 1: Note that, except for the third round, the echoes are not themselves echoed. This however does not influence the later evaluation and decision making, since similarly to the EIG protocol, there is high redundancy in the received information and some of it can be skipped, as pointed out in many previous works including [13]. Moreover, from round to round, the number of nodes in subsequent levels of the constructed EIG tree grows, whereas the new available information (main part of received messages) remains quadratic. This means that all the collected information can be stored in arrays instead of the EIG tree structure. The tree is however used to ease the presentation of the decision making and its analysis.

4.2 Extracting the final decision value

As for the EIG protocol, the decision making consists in assigning new values to the nodes of the EIG tree of each correct process, starting from the leaves (bottom-up evaluation). Let x be a node of the tree and $val(x)$ and $cval(x)$ be the values with which node x is decorated. The evaluation mechanism of the nodes consists in assigning a new value $newval(x)$ to each node x, the new value of the root being the decision value. Let us consider the EIG tree of a correct process p_i.

- When x is a leaf, $newval(x) \leftarrow val(x)$. This will be one of $\top$ or $\perp$.
- When x is an internal node on level l ranging from t down to 1, $newval(x)$ is set to a value v (this is a value equal to $\top$ or $\perp$ for levels greater or equal than 2, and for level 1 it is either a proposed value or $\perp$ if no valid value is received) if there are $(n - t - l)$ children of x the new values of which are set to $\top$ and among them there is a strict majority with the variable $cval$ set to v. More formally:
  (i) Let $T = \{y \mid y \text{ child of } x \wedge newval(y) = \top\}$.
  (ii) $newval(x) \leftarrow v$ if a strict majority of the $cval$ of the children of x that belong to T are set to a same value v.

- When $x = \lambda$, $newval(x) \leftarrow v$ if a strict majority of the new values of its children (these are either proposed values or the default value $\perp$ if no valid value has been computed) are set to a same non-$\perp$ value v; otherwise it is set to a default value v_0.

The intuition that is behind the values $val(x)$, $cval(x)$ and $newval(x)$ of a node x of a given level l is the following. For the nodes of level 1 and level 2, $val(x)$ is a proposed value as in the original EIG protocol. For the other levels $l, 1 < l \leq t$, let us consider that $x = yjk$. $val(x)$ is set to $\top$ if p_k reported that it does not suspect p_j. $cval(x)$ is the echo received from p_k of the value $val(yj)$ (if both p_j and $_k$ are correct $cval(x) = val(yj)$).

The values *newval* are a correction of *val* during the execution of the decision making procedure according to what the child nodes of x reported as an echo of $val(x)$. $newval(x)$ is the information whether p_i considers that p_k suspects p_j or not. $val(x)$ is the suspicion information reported by p_k and $newval(x)$ is the information "computed" by p_i (using the echoes and the suspicion information of the lower levels) on whether p_k suspects p_j or not. Then the values *newval* of this level are used as a mask to re-apply the confirmation mechanism. Only the echoes $cval(xjk)$ of the processes p_k that do not suspect process p_y ($newval(xjk)$ evaluated to $\top$) are considered for the confirmation mechanism of the value $val(xj)$ of node xj (xj is the parent node of node xjk). For a node x of level 1, $newval(x)$ does not contain an information about suspicions as there is only one process id in the label x; it is indeed the corrected value of the original value received by p_i from p_j.

Let us note that whereas in the original EIG protocol only the values associated to the leaves of the tree are used to compute the decision value, i.e., $newval(x)$ for each non-leaf node is a function of the new values assigned to its child nodes, in our protocol the value $newval(x)$ depends on both *newval* and *cval* values associated with its child nodes.

5 Correctness Proof

The correctness proof is tailored along the analysis of the (original) EIG protocol as proposed by Bar-Noy et al. [3] and described in the book by Lynch [17]. Although the steps of both analysis are similar, due to the use of the concept of EIG trees, the proofs often rely on different arguments — more subtle in case of our protocol, as it uses only a small amount of exchanged information. The proof of termination is obvious, therefore in the remainder we focus on validity and agreement.

Lemma 1 *Each message sent by a correct process at round r is confirmed by at least all correct processes at round $r + 1$. Said otherwise, a correct process p_i never suspects a correct process p_j and thus always relays correctly the messages of p_j.*

Proof of Lemma 1: The confirmation mechanism says that an information sent by a process p_j in the main part of a message is confirmed if it is echoed by at least $(n - t)$ different processes; otherwise, the sender is necessarily Byzantine (assuming $t < n/3$). If a correct process p_i sends a message at some round r, it will send the very same message m to all processes and at least all correct processes ($n - t$ processes) will echo the main part of m to all processes at round $r + 1$. Consequently at the end of round $r + 1$ all correct processes will receive at least $n - t$ echos related to message m broadcast at round r by p_i. □

Notation: As each process maintains its own EIG tree structure, there are n nodes labeled x, one for each process of the system. Thus, $val(x)_i$ is used to refer to the value that is associated with the node labeled x of the EIG tree maintained by process p_i.

Lemma 2 *After $t+1$ rounds of the proposed protocol, the following holds. If p_i, p_j and p_k are correct processes, with $i \neq j$, then $val(x)_i = val(x)_j$ for every label x ending in k. Similarly, $cval(x)_i = cval(x)_j$ if x belongs to a level greater than 1 (otherwise, it is not set).*

Proof of Lemma 2: The proof of this lemma follows directly from the fact that a correct process sends the same messages to all processes including itself, and that $val(x)_i$ and $cval(x)_i$ are either proposed values or suspicion information reported to p_i by p_k. □

Lemma 3 extends the previous one to new values.

Lemma 3 *After $t+1$ rounds of the proposed protocol, the following holds. Suppose that x is a label ending with the index of a correct process. Then there is a value v such that $val(x)_i = newval(x)_i = v$ for all correct processes p_i.*

Proof of Lemma 3: The proof is by induction on the length of the tree labels from $t+1$ down to 1.

Base case: the leaves x of the tree (of length $t+1$). From Lemma 2, $val(x)_i$ is the same (call this value v) for all correct processes p_i. And by definition of new values, $newval(x)_i = val(x)_i$ for all correct processes p_i.

Induction: Let us consider a label x of length $\ell, 1 \leq \ell \leq t$ ending with the index of a correct process p_j ($x = yj$). By Lemma 2 all correct processes p_i have the same $val(x)_i$ (call it v). By the proposed algorithm, for any node labeled xk (k being the id of a correct process), $cval(xk)_i = val(x)_i = v$ as $cval(xk)_i$ is the echo of $val(x)_i$ sent by process p_k.

Moreover, by Lemma 1, no correct process suspects a correct process. Consequently, for any node labeled xk (k being the id of a correct process), $val(xk)_i = \top$. By the induction hypothesis, $newval(xk)_i = \top$. Let us now show that a strict majority of the children of node x are correct processes. Indeed, x has $n - \ell$ children. As there are at most t Byzantine processes, we are sure that at least $(n - t - l)$ child nodes xk of x end with the id k of a correct process. As $\ell \leq t$ we have $(n - t - l) \geq (n - 2t) > t$ because $n > 3t$. To sum up, for all child nodes xk of x (p_k being a correct process), we have $newval(xk)_i = \top$ and $cval(xk)_i = val(x)_i$ and these child processes are a strict majority. Consequently, by the decision making procedure of the proposed algorithm, $newval(x)_i$ is set to $val(x)_i$ proving the lemma. (Although for $\ell = 1$ or 2, the values of val and $cval$ are proposed values, the same reasoning holds.) □

Theorem 1 *(Validity) If all correct processes begin with the same initial value v, then the only possible decision value for correct processes is v.*

Proof of Theorem 1: If all correct processes propose the same value v, then $val(j)_i = v$ for any pair p_i, p_j of correct processes. By Lemma 3 $newval(j)_i = v$ for any pair p_i, p_j of correct processes. The majority rule used by the decision procedure implies that $newval(\lambda)_i = v$ for all correct processes p_i. □

In order to prove the agreement property of the protocol, we reuse the definition of path covering set similarly to the EIG protocol. A subset C of the nodes of an EIG tree is a *path covering* if every path from the root to a leaf of the tree contains at least one node in C. Moreover a node x is said to be common for a given execution of the protocol (recall that each process has its own version of the tree) if $newval(x)_i$ is the same for all correct processes p_i. A given path covering is said to be common for an execution if all of its nodes are common in that execution.

The proof of the next Lemma is exactly the same as for the original EIG protocol, however we include it here for completeness.

Lemma 4 *([17]) After $t+1$ rounds of the proposed protocol, there exists a path covering that is common in this execution.*

Proof of Lemma 4: We prove that the set C of all the nodes whose labels are of the form xi where p_i is a correct process is a common path covering. First, by Lemma 3 all these nodes are common as they end with a correct process id. Second, as each path from the root to a leaf has exactly $t+1$ non-root nodes and no process id appears more than once, consequently there is at least one node whose label ends with the id of a correct process since there are at most t Byzantine processes. □

Lemma 5 *After $t+1$ rounds of the proposed protocol, the following holds. If there is a common path covering of the sub-tree rooted at a node labeled x, then either x is common or one of its ancestors is common.*

Proof of Lemma 5: The proof is by induction on the length of the tree labels from $t+1$ down to 1.

Base case: The leaves x of the tree (of length $t+1$). The only node of the common path covering is x itself, which is common by definition of a common path covering.

Induction: Assume the lemma is true for labels greater then l, where $0 \leq l \leq t$, if any. Let x be a node label of size l and C a common path covering rooted at x. If $x \in C$ or if x ends with the id of a correct process then the lemma holds by, respectively, the definition of C and Lemma 3. If x has a correct ancestor then the lemma also holds. So let us consider the case where x is composed of the ids of Byzantine processes. By the definition of a common path covering, any child $x\ell$ of x is a common path covering and, moreover, by the induction hypothesis, ℓ is common. According to the decision making procedure (the computing of $newval(x)$), there are three cases to consider:

- x belongs to level $l = t$. As the process ids that compose a node x of level $l = t$ are all Byzantine (see above), consequently all its child nodes end with the id j of correct processes and these nodes are common by the induction hypothesis. On the other side, for each child node xj, $cval(xj)$ is a value reported by a correct process p_j. By Lemma 1, correct processes report correctly messages and thus, as the decision making procedure is deterministic and correct processes have the same data concerning x, all correct processes will compute the same $newval(x)$ and x is common.

- x belongs to level $l \leq t-1$. Suppose $x = yj$, for some node y and process p_j. Let us consider the set of process ids $top\_set = \{k \mid xk \text{ is a child node of } x \wedge newval(xk) = \top\}$. This set

contains process ids k such that the child node of x the label of which terminates with the id k has a newval set to $\top$. This set is the same for all the EIG trees maintained by the different correct processes by the induction hypothesis. If $\mid top\_set \mid < (n-t-l)$ then by the decision making procedure, any correct process p_i sets $newval(x)$ to the same value $\perp$ and the lemma follows.

If $\mid top\_set \mid \geq (n-t-l)$ then at least $(n-t-l)-(t-l) = n-2t > t+1$ of the processes represented in *top_set* are correct (recall that the number of Byzantine processes whose ids are in *top_set* is at most $t-l$ as x is composed of the ids of l Byzantine processes). This means that among the processes represented in the set *top_set* there is a strict majority of correct processes. Moreover, this majority of such correct processes p_k do not suspect process p_j (recall that $x = yj$) by the definition of the set *top_set* and by Lemma 3 applied to node xk. If they do not suspect p_j this means that they reported the same value $cval(xk)$. Indeed, if two correct processes report different values from some process, it is necessary that at most one of these processes can confirm the value it reported and the second one will suspect the sender (by the confirmation mechanism). Hence, the value $cval(xk)$ is a strict majority among the processes of *top_set*, and consequently this same value is assigned to $newval(x)$ and x is common.

- $x = \lambda$ (level $l = 0$). By the definition of $newval(x)$ and the induction hypothesis, x is common. In more details, suppose to the contrary that λ is not common. Then, by inductive hypothesis, each of its children y consisting of a unique process id would be common, which means that it has the same $newval(y)_k$ across correct processes p_k. Consequently, by the definition of $newval(\lambda)$ and by the fact that the number of children of λ consisting of a correct process id is at least $n - t > t$, the $newval(\lambda)_i$ are the same across correct processes p_i. This means that λ is common, violating our contradictory assumption.

□

Theorem 2 *(Agreement) The proposed protocol ensures the agreement property of consensus.*

Proof of Theorem 2: After $t+1$ rounds of the algorithm, by lemmas 4 and 5 and the fact that the node λ has no ancestor, λ is common. Agreement follows as $newval(\lambda)$ is the decision value. □

6 Complexity Analysis and Improvement

If we consider binary consensus (only two possible value can be proposed), the main algorithm, as described in Section 4, achieves $O(n^3)$ communication complexity, in terms of the number of bits sent by any correct process. Each process sends, respectively, $O(1)$, $O(n)$ and $O(n^2)$ bits during the first three rounds, and then $O(n^2)$ bits in each subsequent round: a list of suspected processes and the echoes of the lists received in the previous round (n^2 bits at most). For the maximal value of t (which is $O(n)$), the communication bit complexity of this protocol is $O(n^3)$ per process, and thus $O(n^4)$ for all correct processes in total.

In the case of multivalued consensus, the proposed value of each process is taken from a set V of proposable values, the size of which is k. The bit complexity of the first three rounds becomes

$O(n^2) \log_2 k$ per process, where the factor $\log_2 k$ comes from the size of the binary representation of a proposed value. The complexity of the subsequent rounds of the algorithm does not depend on k, as processes only exchange suspicion information, namely, $O(n^3)$ bits per process. The total bit complexity is thus $O(\max(n^4, n^3 \log k))$. If $\log_2 k = O(n)$, the bit complexity of the algorithm is $O(n^4)$.

However, similarly to the floodset protocol for solving the consensus problem in crash-prone synchronous systems, our protocol can be modified to use only an incremental dissemination of the suspicions (instead of sending the whole list of suspected processes all the time). Indeed, once a process suspects another process it will suspect it until the end of the execution and there is no need to resend this information at each round. Namely, each suspicion is sent only once, that is, in the first time where it is discovered. A correct process can suspect at most t processes. Obviously, if a suspicion is sent only once, the echo is also sent only once by correct processes. A Byzantine process can be suspected by possibly all the other processes. Due to the relay, we can say that during the execution of the protocol each process sends t suspicions and relays $(n-1)t$ suspicions. Each relay of a suspicion needs two process ids, i.e., who suspects who. That is, the protocol needs $O(nt \log n)$ bits per process, where the factor $\log_2 n$ comes from the size of the binary representation of a process id. Moreover, each process sends respectively $O(1)$, $O(n)$ and $O(n^2)$ bits during the first three rounds, which are special rounds. Finally, the property of well-formed messages requires from each process to send its id, of logarithmic size, to each process in every round; this contributes another $O(t \log n)$ to the bit communication complexity per each process. For the maximal value of t, which is $O(n)$, the bit communication complexity of the modified protocol is $O(n^2 \log n)$ per process, therefore $O(n^3 \log n)$ in total for binary consensus.

Again, if we consider that the proposed values are taken from a set V of size k then the complexity of the algorithm becomes $O(n^3 \log(\max(n, k)))$.

Acknowledgments

We would like to thank Nancy Lynch for suggesting this research direction when Achour Mostefaoui spent three months visiting her group. We are also thankful to Yoram Moses for all the discussions and valuable remarks. Finally we would like to thank the anonymous reviewers whose comments helped to improve this paper.

References

[1] Aspnes J., Faster randomized consensus with an oblivious adversary. *Proc. 31st ACM Symposium on Principles of Distributed Computing (PODC 2012)*, pp. 1-8, 2012.

[2] Bar-Noy A., Dolev D., Families of Consensus Algorithms. *Proc. 3rd Aegean Workshop on Computing (AWOC 1988)*, pp. 380-390 1988.

[3] Bar-Noy A., Dolev D., Dwork C., and Strong H.R., Shifting Gears: Changing Algorithms on the Fly To Expedite Byzantine Agreement. *Proc. 6th ACM Symposium on Principles of Distributed Computing (PODC 1987)*, pp. 42-51 1987.

[4] Berman P., Garay J., Efficient Distributed Consensus with $n = (3+\epsilon)t$ Processors. *5th International Workshop on Distributed Algorithms (WDAG'91)*, Delphi, Greece, LNCS 579 Springer, pp. 129-142, 1991.

[5] Berman P., Garay J., Perry K., Towards Optimal Distributed Consensus (Extended Abstract). *Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS 1989)*, pp. 410-415 1989. FOCS 1989: 410-415.

[6] Bracha G. and Toueg S., Resilient Consensus Protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC 1983)*, ACM Press, Montreal.

[7] Coan B., A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols. *Proc. 5th ACM Symposium on Principles of Distributed Computing (PODC 1986)*, pp. 63-72, 1986.

[8] Chlebus B.S., Kowalski D.R., and Strojnowski M., Fast Scalable Deterministic Consensus for Crash Failures. *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC 2009)*, pp. 111-120, 2009.

[9] Dolev D., Fischer M., Fowler R., Lynch N., Strong R., An Efficient Algorithm for Byzantine Agreement without Authentication. *Information and Control*, vol 52(3):257-274, 1982.

[10] Dolev D. and Reischuk R., Bounds on Information Exchange for Byzantine Agreement. *Journal of the ACM* 32(1): 191-204, 1985.

[11] Dolev D., Reischuk R., Strong R., Early Stopping in Byzantine Agreement. *Journal of the ACM*, vol 37(4):720-741, 1990.

[12] Fisher M. and Lynch N., A lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 14: 183-186, 1982.

[13] Garay J.A. and Moses Y., Fully polynomial Byzantine agreement in $t + 1$ rounds. *Proc. 25th ACM Symposium on Theory of Computing (STOC 1993)*, ACM Press, pp. 31-41, 1993.

[14] Garay J.A. and Moses Y., Fully Polynomial Byzantine Agreement for $n > 3t$ Processors in $t + 1$ Rounds. *SIAM Journal on Computing*, vol 27(1):247-290, 1998.

[15] King V. and Saia J., Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary. *Proc. 29th ACM Symposium on Principles of Distributed Computing (PODC 2010)*, pp. 420-429, 2010.

[16] Lamport L., Shostak R., and Pease M., The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4: 382-401, 1982.

[17] Lynch N.A., Distributed Algorithms. *Morgan Kaufmann*, 1996.

[18] Moses Y., Waarts O., Coordinated Traversal: $(t + 1)$-Round Byzantine Agreement in Polynomial Time. *29th IEEE Symposium on Foundations of Computer Science (FOCS'88)*, pp. 246-255, 1988.

[19] Pease L., Shostak R., and Lamport L., Reaching Agreement in Presence of Faults. *Journal of the ACM*, 27(2):228-234, 1980.

[20] Toueg S., Perry K., Srikanth T., Fast Distributed Agreement. *SIAM Journal of Computing*, vol 16(3):445-457, 1987.